Multi-agent active learning for distributed black-box optimization

Loris Cannelli, *Member, IEEE*, Mengjia Zhu, *Member, IEEE*, Francesco Farina, Alberto Bemporad, *Fellow, IEEE*, Dario Piga, *Member, IEEE*

***Abstract*— This paper proposes a method to solve global optimization problems over a multi-agent network, where the objective function, possibly subject to global constraints, is not analytically known, but can only be evaluated at any query point. It is assumed that the cost function to be minimized is the sum of local cost functions, each of which can be evaluated by the associated agent only. The proposed algorithm asks the agents at each iteration first to fit a *surrogate function* to local samples, and subsequently to minimize, in a cooperative fashion, an *acquisition function*, in order to generate new samples to query. In this paper we build the acquisition function as the sum of the local surrogates, in order to exploit the knowledge of these estimates, plus another term that drives the minimization procedure towards unexplored regions of the feasible space, where better values of the objective function might be present. The proposed scheme is a distributed version of the existing algorithm GLIS (GLobal optimization based on Inverse distance weighting and Surrogate radial basis functions), and share with it the same low-complexity and competitiveness, with respect to, for instance, Bayesian optimization. Experimental results on benchmark problems and on distributed calibration of Model Predictive Controllers (MPC) for autonomous driving applications demonstrate the effectiveness of the proposed method.**

***Index Terms*— Black-box optimization, distributed optimization, Model Predictive Control, multi-agent networks, surrogate models**

I. Introduction

Active learning algorithms for black-box global optimization aim to minimize (or, equivalently, maximize) an expensive-to-evaluate objective function $f(x)$, whose analytical expression is typically not available and can only be evaluated through experiments or simulations. Different active learning algorithms for black-box optimization with a minimum amount of function evaluations have been proposed in the last decades. The most popular method is *Bayesian Optimization* (BO) [1], which relies on successively constructing a probabilistic surrogate function (typically, a Gaussian Process) approximating the objective $f(x)$. The surrogate is then used at each iteration of BO to select the next point where to query $f(x)$, by trading-off exploitation (searching for values of x where $f(x)$ is expected to be optimal) and exploration (searching for points x for which $f(x)$ is highly uncertain). The same rationale is also adopted by other active-learning based approaches for black-box global optimization [2], [3], [4], [5], [6]. Examples and applications of active learning for black-box global optimization include material and drug design [7], [8]; calibration of controller parameters [9], [10]; tuning of hyper-parameters for machine learning algorithms and choice of neural network architectures in deep learning [11], [12], just to cite a few.

In this paper, we develop a distributed version of the GLIS algorithm, recently developed by one of the authors in [5]. The need of distributed active learning algorithms for black-box optimization comes from the recent research interest in networked multi-agent systems [13], [14], [15]. Network-structured problems can be found in several engineering areas, including swarm robotics (e.g., distributed learning, flock control), distributed machine learning (e.g., logistic regression, dictionary learning, tensor factorization), networked information processing (e.g., graph signal processing, parameters estimation, detection, and localization), communication networks (e.g., resource allocation in multi-cellular systems), sensor networks, data-based networks (i.e., Facebook, Twitter, Google), and other areas. The commonality of such applications is that they need to perform a decentralized optimization; this happens mainly because of two aspects: i) a lack of a central controller/authority, and ii) an inherent time-varying dynamics of the connectivity structure. Indeed, in several applications the presence of a central controller (a master node) is impractical or unattractive for various reasons: 1) its resources may be insufficient to coordinate the whole network and communicate with all the agents (e.g., limited bandwidth); 2)

Manuscript received October 14, 2022.

Loris Cannelli and Dario Piga are with the IDSIA Dalle Molle Institute for Artificial Intelligence, SUPSI-USI, 6962, Lugano, Switzerland (e-mail:{loris.cannelli, dario.piga}@supsi.ch).

Mengjia Zhu and Alberto Bemporad are with IMT School for Advanced Studies, Lucca, Italy (e-mail: {mengjia.zhu, alberto.bemporad}@imtlucca.it).

Francesco Farina is with Moonfire Ventures (e-mail: francesco@moonfire.com).

its single failure will cause the entire system to fail (robustness concerns); 3) privacy and confidentiality of the local information of the agents can not be preserved; 4) some of the agents in the network might be low-power devices that can communicate only with nodes in their physical proximity, making unfeasible the existence of a star topology (or a spanning tree). Furthermore, additional advantages of distributed systems is their inherent flexibility: the network topology and connectivity may easily be time-varying due, e.g., to agents mobility, link failures, or power outage.

The main idea of GLIS is to recursively build a surrogate of the objective function $f(x)$ as a linear combination of a *Radial Basis Function* (RBF), whose parameters are estimated by quadratic programming. The acquisition function used to select the query point x at each iteration of GLIS is then constructed as a weighted sum of the surrogate and of an *Inverse Distance Weighting* (IDW) function that is used to promote exploration of the input space. In the distributed version of GLIS presented in this paper, called D-GLIS, we consider a distributed black-box optimization problem in which N agents attempt optimizing a global separable objective given by the sum of local objectives $f_i(x)$, i.e., $f(x) = \sum_{i=1}^{N} f_i(x)$. Following the GLIS approach, at each iteration of D-GLIS, the i-th agent updates a surrogate $\hat{f}_i$ of its local objective f_i based on its current estimate of the surrogate $\hat{f}_i$. Then, each agent optimizes a local acquisition function to find the next point to query locally. The local acquisition function consists of the combination of a surrogate of the global objective $f(x)$ and a local IDW function promoting the exploration of areas not yet explored by the i-th agent.

In this paper, we refer to distributed black-box optimization in terms of:

- the experiments performed by the agents. In fact, the input space is cooperatively explored by the agents, with the final goal of minimizing the global objective $f(x)$. Unlike other approaches for multi-agent learning [16], [17], [18], [19], we assume that the agents do not share information regarding their local objective f_i nor the corresponding surrogate $\hat{f}_i$;
- the optimization of the acquisition function minimized by each agent to select its query point x where to evaluate $f_i(x)$. This requires to optimize a local acquisition function which includes the global surrogate $\sum_{i=1}^{N} \hat{f}_i(x)$. Since each agent builds its own surrogate, and this information is not shared among the agents, D-GLIS relies on decentralized consensus/optimization schemes. This allows the agents to cooperate over a network in order to achieve a global performance objective (e.g., the global minimization of $f(x)$), by exchanging a limited amount of local information with their one-hop neighbors only.

The paper is organized as follows. Section II describes the problem formulation and summarizes the fundamental assumptions. Section III presents the proposed distributed learning scheme D-GLIS. Section IV discusses the building blocks of D-GLIS that lie in the field of distributed optimization. Section V presents numerical results obtained in applying D-GLIS for solving a set of benchmark global optimization problems in a distributed way, and for the decentralized calibration of an MPC for autonomous driving. Finally, some conclusions and possible future directions are drawn in Section VI.

II. Problem formulation

Consider a network composed by N computing units (agents). We address the problem of solving the following optimization problem:

$$x^\star = \arg\min_{x \in \mathcal{X}} f(x), \tag{1}$$

where $x \in \mathbb{R}^n$ and $\mathcal{X} \subseteq \mathbb{R}^n$ is the set of feasibility which is supposed to be known. We assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is separable:

$$f(x) \triangleq \sum_{i=1}^{N} f_i(x), \tag{2}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ is the local cost function of the i-th agent. We suppose that:

- the analytical expression of the local functions f_i (with $i = 1, \ldots, N$) and their corresponding gradients are not available, and f_i can only be observed by the i-th agent through the evaluation of $f_i(x)$ at any selected $x \in \mathcal{X}$, possibly in a noisy way. More specifically, the agent measures $y = f_i(x) + \epsilon$ of $f_i(x)$, where ϵ is an unknown error that can change from one evaluation to another;
- each agent performs its own function evaluation $f_i(x)$, without sharing this information with the other agents. At the same time, the agents cooperate to meet the global objective of solving problem (1);
- evaluating $f_i(x)$ is expensive, and thus problem (1) needs to be solved within a limited number of function evaluations.

As an example, $f(x)$ can be a global performance index, which is given by the average of the local performance indexes $f_i(x)$ of each agent. The agents might not provide information on the local function evaluations $f_i(x)$ they performed for several reasons, such as for privacy issues, reduction of communication costs, lack of a central coordinator (master), etc.

The communication among the agents is modeled as a fixed, directed weighted graph $\mathcal{G} = (\{1, \ldots, N\}, \mathcal{E}, \mathcal{A})$, where $\{1, \ldots, N\}$ is the set of the vertices/agents, $\mathcal{E} \subseteq \{1, \ldots, N\} \times \{1, \ldots, N\}$

is the set of edges/communication links, and $\mathcal{A} \in \mathbb{R}^{N\times N}$ is the weighted adjacency matrix of the graph. The edge $(i,j) \in \mathcal{E}$ models the fact that agent i can send a message to agent j. $\mathcal{A}$ is compliant with the topology described by $\mathcal{E}$, that is to say, being α_{ij} the $(i,j)-$entry of $\mathcal{A}$, then $\alpha_{ij} > 0$ if $(i,j) \in \mathcal{E}$, and $\alpha_{ij} = 0$ otherwise. In the rest of this work it will be assumed that i) $\mathcal{G}$ is strongly-connected, and ii) $\mathcal{A}$ is doubly stochastic. These are two common minimal assumptions (see, for example, [20], [21]) ensuring that the information of each agent influences the information of any other agent infinitely often in time (connectivity). Finally, we denote by $\mathcal{N}_i$ the set of *in-neighbors* of node i in the fixed graph $\mathcal{G}$, i.e., $\mathcal{N}_i \triangleq \{j \in \{1,\dots,N\}\,|(j,i)\in\mathcal{E}\}$. The proposed work can easily be adapted to work on undirected graphs too.

In the following section, we describe a distributed variant of the GLIS algorithm [5], called D-GLIS, to solve the global problem (1) through a decentralized strategy relying on the communication graph $\mathcal{G}$ and based on active learning.

III. D-GLIS

We propose an algorithm to find the optimal solution $x^\star$ of problem (1) in an iterative way, through the cooperation of the N agents. At each iteration of the algorithm, one agent, selected cyclically according to a round-robin scheme, selects the new point x where to perform the function evaluation $f_i(x)$, with the cooperation of all the other agents. The main features of D-GLIS, which differentiate it from GLIS, are the following:

- at each iteration, one agent, selected in a cyclic way, say the i-th agent, constructs a local acquisition function $a_i : \mathbb{R}^n \to \mathbb{R}$ in order to compute the next point x where to evaluate its own local function $f_i(x)$, using local information extracted from the dataset $\mathcal{D}_i$ and global information shared through the communication network;
- the optimization of the aforementioned local acquisition functions a_i requires cooperation among the agents, in order to collect, at the same time, information regarding the global cost function $f(x)$, and it is thus performed in a distributed way, exchanging only limited amount of data among the agents.

A. Local surrogate functions

Assume that each agent i has collected a local dataset $\mathcal{D}_i = \{x_j, y_j\}_{j=1}^{M_i}$ of length M_i, where y_j is the noisy observation of $f(x_j)$. Because of the assumptions above, $\mathcal{D}_i$ is not shared with the other agents.

Among many possible choices to construct the surrogate $\hat{f}_i$ approximating the true unknown local objective f_i for each agent $i = 1,\dots,N$, we adopt the following weighted linear combination of RBFs [2], [22], according to the original version of GLIS [23]:

$$\hat{f}_i(x) = \sum_{k=1}^{M_i} \beta_k^{(i)} \phi(\epsilon d(x, x_k)), \tag{3a}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is an RBF, with $d(x, x_k)$ being any distance function between x and x_k and $\epsilon > 0$ a scalar hyper-parameter defining the shape of the RBF. The unknown coefficients $\beta^{(i)} = [\beta_1^{(i)} \ \dots \ \beta_{M_i}^{(i)}]'$ are determined by fitting the model $\hat{f}_i(x)$ to the dataset $\mathcal{D}_i$ in order to minimize the regularized squared error:

$$\beta^{(i)} = \arg\min_{\beta^{(i)}} \sum_{k=1}^{M_i} \left\| y_k - \sum_{k=1}^{M_i} \beta_k^{(i)} \phi(\epsilon d(x, x_k)) \right\|^2 \tag{3b}$$

$$+ \gamma \left\| \beta_k^{(i)} \right\|^2 . \tag{3c}$$

Some RBFs commonly used are $\phi(\epsilon d) = \frac{1}{1+(\epsilon d)^2}$ (*inverse quadratic*), $\phi(\epsilon d) = e^{-(\epsilon d)^2}$ (*squared exponential kernel*), and $\phi(\epsilon d) = (\epsilon d)^2 \log(\epsilon d)$ (*thin plate spline*), with hyper-parameter ϵ tuned through cross-validation.

Once local surrogate functions $\hat{f}_i$ are estimated, the surrogate $\hat{f}$ of the global objective f is simply given by:

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{f}_i(x). \tag{4}$$

If the global surrogate $\hat{f}$ were known to all agents, it could be in principle minimized in order to find the new sample x_{T+1} at iteration $T+1$ of D-GLIS. However, two issues arise. First, we assume that the agents do not share their local surrogates $\hat{f}_i$. Thus, distributed algorithms must be adopted to minimize $\hat{f}$. This point will be discussed in Section IV. The second issue is due to the fact that, by considering only the surrogate $\hat{f}$, we only exploit the current available observations. Thus, the global minimum of problem (1) can be missed as the surrogate is not guaranteed to well approximate the true objective f in unexplored regions of the input domain $\mathcal{X}$. Therefore, a term promoting exploration of the input space $\mathcal{X}$ must be considered, which should be different for each agent, as the agents may explore the input space in different ways. This point is discussed in the following paragraph.

B. Local inverse distance weighting functions

According to [5], the inverse distance weighting (IDW) function is used to promote exploration. In particular, for the i-th agent, the IDW function is defined as

$$z_i(x) = \begin{cases} 0 & x \in \{x_1, \dots, x_{M_i}\} \\ \tan^{-1}\left(\frac{1}{\sum_{j=1}^{M_i} w_j(x)}\right) & \text{otherwise} \end{cases} \tag{5}$$

where $w_j(x) = \frac{1}{\|x-x_j\|^2}$ and $x_1, \dots, x_{M_i}$ are the datapoints contained in $\mathcal{D}_i$. Clearly, $z(x) = 0$ for all inputs already tested by the agent and $z(x) > 0$ in $\mathbb{R}^n \setminus \{x_1, \dots, x_{M_i}\}$. Furthermore, the value of z_i increases as the (sum of the) distances between x and the already tested inputs $\{x_1, \dots, x_{M_i}\}$ increases.

C. Local acquisition functions

Given an exploration hyper-parameter $\delta \geq 0$, the local *acquisition function* $a_i : \mathcal{X} \to \mathbb{R}$ is constructed in order to balance exploration and exploitation. Specifically, given the global surrogate $\hat{f}$ (unknown to agent i), define a_i as follows:

$$a_i(x) \triangleq \frac{\hat{f}(x)}{\Delta\hat{f}} - \delta z_i(x), \tag{6}$$

where $\Delta\hat{f}$ is the range of the surrogate $\hat{f}$, i.e.,

$$\Delta\hat{f} \triangleq \max_{x\in\mathcal{X}} \hat{f}(x) - \min_{x\in\mathcal{X}} \hat{f}(x)$$

and is used in (6) as a normalization factor to ease the selection of the exploration parameter $\delta \in (0, 1]$.

At each iteration of D-GLIS, one agent, say agent i, $i = 1, \dots, N$, is selected in a round-robin fashion and the input parameter $x^{i,*}$ to test for this agent is computed, according to GLIS, as the solution of the optimization problem:

$$x^{i,*} = \arg\min_{x\in\mathcal{X}} a_i(x). \tag{7}$$

It is worth noticing that in constructing the acquisition function a_i in (6) (or equivalently, in selecting the next point to test) the global surrogate $\hat{f}$ is considered by the i-th agent, while exploration is driven by a local IDW z_i. This is because the agents should cooperate to optimize the global objective f, while each agent should also evaluate its local objective f_i through its own exploration of the input domain $\mathcal{X}$.

D. Iterative optimization

Once the new input $x^{i,*}$ is selected, i) the i-th agent evaluates $y_i = f_i(x^{i,*}) + \epsilon$, ii) the local surrogate $\hat{f}_i$ and IDW functions z_i are updated, iii) a new agent j is selected. This procedure is iterated until a maximum number of iterations $T_{\max}$ is reached. Finally, the optimal solution $x^\star$ is computed by only minimizing the final global objective $\hat{f}$, thus switching off the exploration term. The main steps of the D-GLIS approach are summarized in Algorithm 1.

Algorithm 1 D-GLIS

Inputs: maximum number of function evaluations per agent $N_{\max}$; exploration parameter δ; constraint set $\mathcal{X}$; initial datasets $\mathcal{D}_i = \{x_j, y_i\}_{j=1}^{M_i}$, and initial surrogate functions $\hat{f}_i$ -constructed from $\mathcal{D}_i$- for all agents $i = 1, \dots, N$.

```
1:  repeat
2:      select agent $i$ according to a cyclic round-robin rule
3:      build the IDW function $z_i$ in (5) from $\{x_j\}_{j=1}^{M_i}$
4:      define the local acquisition function $a_i$ in (6)
5:      compute $x^{i,*} = \arg\min_{x\in\mathcal{X}} a_i(x)$ via distributed optimization (Section IV)
6:      evaluate $y^{i,*} = f(x^{i,*}) + \epsilon$
7:      update the local dataset $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{x^{i,*}, y^{i,*}\}$
8:      $M_i \leftarrow M_i + 1$
9:      estimate a new local surrogate function $\hat{f}_i(x)$ based on $\mathcal{D}_i$
10:     construct the global surrogate $\hat{f}(x)$ as in (4)
11: until maximum numbers of iterations $T_{\max} = NN_{\max}$ is reached
12: compute consensus $x^* = \arg\min_x \sum_{i=1}^N \hat{f}_i(x)$ via distributed optimization (Section IV)
```

Output: consensus $x^\star$.

Remark The D-GLIS implementation described in Algorithm 1 is a *sequential* version, meaning that at each iteration one agent i is selected in a cyclic way for computing its acquisition function a_i and for consequently updating its dictionary $\mathcal{D}_i$. Different implementations are possible, for example it is possible to consider a *parallel* version where all the agents compute at each iteration their acquisition functions, solve in a distributed way at the same time N optimization problems as (7), and finally update their datasets $\mathcal{D}_i$ with the just computed new points.

IV. Distributed optimization

In the D-GLIS algorithm described in the previous section, the generic agent i optimizes its own acquisition function $a_i(x)$, which also depends on the local surrogate functions $\hat{f}_j$ (with $j \neq i$) of the other agents. However, local surrogates of the other agents are assumed not to be known by the i-th agent. This requires to use distributed algorithms, where all the agents communicate with the others to optimize the local acquisition function a_i of the i-th agent.

In order to solve (7) in a cooperative fashion, the agents leverage the GTAdam [24] distributed algorithm. GTAdam is a distributed version of the popular Adam algorithm [25]. Adam is a gradient-like optimization scheme that solves problems in the form of (7) in a *centralized* way. At each iteration $k \in \mathbb{N}_+$ of Adam, a solution estimate x^k is updated by computing a descent direction which is properly adjusted using the gradient history. More specifically, an estimate of the mean and of the variance of the gradient sequence $\left\{\nabla a_i(x^k)\right\}_{k\in\mathbb{N}_+}$ (i.e., the first and the second momentum of the gradient sequence), respectively denoted as m^k and v^k, are computed at each iteration k, and properly combined in order to obtain the descent direction for the update of x^k. The use of the momenta m^k and v^k are very effective in making Adam a fast optimization method [25], [26], [27].

A pseudo-code of Adam is shown in Algorithm 2, with $\odot$ denoting the Hadamard product, $\Pi_{\mathcal{X}}(\cdot)$ the Euclidean projection onto the constraint set $\mathcal{X}$, and $\frac{m^k}{\sqrt{v^k+\epsilon}}$ is an element-wise ratio.

Algorithm 2 Adam algorithm for problem (7)

Inputs: step-size $\alpha > 0$; hyperparameters $\beta_1, \beta_2 \in (0,1)$; constraint set $\mathcal{X}$; iteration counter $k = 0$; random starting point $x^0 \in \mathcal{X}$; standard initialization $m^0 = v^0 = 0$, $g^0 = \nabla a_i(x^0)$.

```
1: repeat
2:     compute the first momentum $m^{k+1} = \beta_1 m^k + (1-\beta_1) g^k$
3:     compute the second momentum $v^{k+1} = \beta_2 v^k + (1-\beta_2) g^k \odot g^k$
4:     update the estimate solution $x^{k+1} = x^k - \alpha \frac{m^{k+1}\sqrt{1-\beta_2}}{(1-\beta_1)\sqrt{v^{k+1}+\epsilon}}$
5:     ensure that the estimate solution is feasible $x^{k+1} = \Pi_{\mathcal{X}}(x^{k+1})$
6:     compute the gradient vector $g^{k+1} = \nabla a_i(x^{k+1})$
7:     update the iteration counter $k \leftarrow k+1$
8: until maximum number of iterations is reached
```

Output: solution estimate $x^{i,*} = x^k$.

GTAdam [24] is the distributed version of Adam, that has the purpose of solving problems in the form of (7) over a network of agents by means of local computation and communication only, without any central coordinator. Furthermore, in this distributed setting it is assumed that $\hat{f}$ (which is one of the term in (7)) is not globally known, but each agent $i = 1, \ldots, N$ only knows $\hat{f}_i$. In order to make Adam distributed, the renowned *gradient tracking* algorithm [28], described next, is encapsulated into the Adam framework. In the gradient tracking algorithm, each agent i, at any iteration k, maintains and updates two local states s_i^k and x_i^k. While s_i^k is an estimate of the gradient of the whole function to be minimized, which is updated at each iteration according to a dynamic tracking mechanism, x_i^k is a solution estimate, which is updated at each iteration firstly by moving along the estimated gradient direction s_i^k, and secondly by performing a consensus step to force asymptotic agreement among the solution estimates of all agents. It is important to note that, according to the gradient tracking algorithm, the only communication requirement is, for each agent, to transmit s_i^k and x_i^k to its own out-neighbors at each iteration. A pseudo-code of the gradient tracking algorithm is shown in Algorithm 3 from the perspective of agent i only; it is here assumed that an estimate for $\Delta\hat{f}$ is available, and it will be described later in this section how to compute it. Note that the pseudo-code of Algorithm 3 from the perspective of an agent $j \neq i$ remains the same, except for the fact that the local gradient $g_j^k = \frac{1}{\Delta\hat{f}} \nabla \hat{f}_j(x_j^k)$ does not include the IDW function z_i.

Algorithm 3 Gradient tracking for problem (7) (from the perspective of agent i)

Inputs: stepsize $\alpha > 0$; normalization factor $\Delta\hat{f} > 0$; exploration parameter $\delta \in (0,1]$; constraint set $\mathcal{X}$; iteration counter $k = 0$; random starting point $x_i^0 \in \mathcal{X}$; standard initialization $s_i^0 = g_i^0 = \nabla\left(\frac{\hat{f}_i(x_i^0)}{\Delta\hat{f}} - \delta z_i(x_i^0)\right)$.

```
1: repeat
2:     update the estimate solution $x_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} x_j^k - \alpha s_i^k$
3:     ensure that the estimate solution is feasible $x_i^{k+1} = \Pi_{\mathcal{X}}(x_i^{k+1})$
4:     compute the local gradient vector $g_i^{k+1} = \nabla\left(\frac{\hat{f}_i(x_i^{k+1})}{\Delta\hat{f}} - \delta z_i(x_i^{k+1})\right)$
5:     update the gradient estimate $s_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij} s_j^k + g_i^{k+1} - g_i^k$
6:     update the iteration counter $k \leftarrow k+1$
7: until maximum numbers of iterations is reached
```

Output: solution estimate $x^{i,*} = x_i^k$.

GTAdam combines Algorithm 2 with Algorithm 3 for solving (7) in a distributed fashion. Again, each agent i knows only a local part $\hat{f}_i$ of the sum-utility $\hat{f}$ and can communicate with its out-neighbors only. In GTAdam each agent maintains four local states: i) x_i^k, a local estimate of the current optimal solution $x^{i,*}$; ii) s_i^k, a local estimate of the gradient of the whole cost function a_i; iii) m_i^k, an estimate of first momentum of s_i^k; and iv) v_i^k, an estimate of the second momentum of s_i^k. The agents need to communicate among each other only s_k^i and x_i^k at each iteration in order to have convergence to the problem solution. Finally, each agent can estimate the normalization factor $\Delta\hat{f}$ as $\Delta\hat{f}_i \triangleq \max_{x\in\mathcal{D}_i} \hat{f}_i(x) - \min_{x\in\mathcal{D}_i} \hat{f}_i(x)$. A pseudo-code of GTAdam is shown in Algorithm 4 from the perspective of agent i only (as for Algorithm 3, the pseudo-code of Algorithm 4 from the perspective of an agent $j \neq i$ remains the same, except for the fact that the local gradient $g_j^k = \frac{1}{\Delta\hat{f}_j} \nabla \hat{f}_j(x_j^k)$ does not include the IDW function z_i).

Algorithm 4 GTAdam for problem (7) (from the perspective of agent i)

```
Inputs: stepsize $\alpha > 0$; exploration parameter $\delta \in (0,1]$; constraint set $\mathcal{X}$; constants for numerical robustness $\epsilon \in (0,1)$, $G > 0$; iteration counter $k = 0$; random starting point $x_i^0 \in \mathcal{X}$; standard initialization $m_i^0 = v_i^0 = 0$, $s_i^0 = g_i^0 = \nabla\left(\frac{\hat{f}_i(x_i^0)}{\Delta\hat{f}_i} - \delta z_i(x_i^0)\right)$.
1: repeat
2:     compute the first momentum $m_i^{k+1} = \beta_1 m_i^k + (1-\beta_1)s_i^k$
3:     compute the second momentum $v_i^{k+1} = \min\left\{\beta_2 v_i^k + (1-\beta_2)s_i^k \odot s_i^k, G\right\}$
4:     update the estimate solution $x_i^{k+1} = \sum_{j\in\mathcal{N}_i} a_{ij}x_j^k - \alpha\frac{m_i^{k+1}}{\sqrt{v_i^{k+1}+\epsilon}}$
5:     ensure that the estimate solution is feasible $x_i^{k+1} = \Pi_{\mathcal{X}}(x_i^{k+1})$
6:     compute the local gradient vector $g_i^{k+1} = \nabla\left(\frac{\hat{f}_i(x_i^{k+1})}{\Delta\hat{f}_i} - \delta z_i(x_i^{k+1})\right)$
7:     update the gradient estimate $s_i^{k+1} = \sum_{\mathcal{N}_i} a_{ij}s_j^k + g_i^{k+1} - g_i^k$
8:     update the iteration counter $k \leftarrow k+1$
9: until maximum numbers of iterations is reached
Output: solution estimate $x^{i,*} = x_i^k$.
```

The strong connectivity assumption on $\mathcal{G}$ (mentioned in Section II) ensures that eventually the information flow of each agent can reach any other agent in the network, while the double stochasticity of $\mathcal{A}$ guarantees that the agents asymptotically will converge to the same stationary point (see [24] for more details).

Note that GTAdam in D-GLIS is implemented not only in `Step 5` for solving (7) as described above, but, similarly, in `Step 12` too. The distributed optimization problem in `Step 12` is the same appearing in `Step 5` with the difference that the IDWs do not appear.

V. Examples

In this section we test the D-GLIS algorithm on standard benchmark global optimization problems (Section V-A) and distributed design of model predictive control for autonomous driving (Section V-B).

All our experiments were run on an Intel i7-8665u, 1.9GHz processor, 32GB RAM, no GPUs. Codes of D-GLIS are in Python and can be downloaded at `https://leon.idsia.ch/lib_download`. The open-source *disropt library* [29] is used for the distributed operations.

A. Benchmark optimization problems

We test D-GLIS on four standard benchmark global optimization problems, denoted as `brent`, `camelsixumps`, `hartman3` and `ls`. Problems `brent`, `camelsixumps`, and `hartman3` problems are defined in [30] and `ls` is a distributed least square problem described in the following:

- `brent`: number of agents $N = 3$; $x \in \mathbb{R}^2$; cost function:
$$f(x) = \underbrace{(x_1+10)^2}_{f_1(x)} + \underbrace{(x_2+10)^2}_{f_2(x)} + \underbrace{e^{-x_1^2-x_2^2}}_{f_3(x)}; \tag{8}$$
constraints $x_i \in [-10 \quad 10]$ $(i = 1,2)$; global optimizer $x^\star = [-10 \quad -10]^\top$; global optimum $f(x^\star) \simeq 0$.
- `camelsixumps`: number of agents $N = 3$; $x \in \mathbb{R}^2$; cost function:
$$f(x) = \underbrace{(4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2}_{f_1(x)} + \underbrace{x_1x_2}_{f_2(x)} \tag{9}$$
$$+ \underbrace{(4x_2^2-4)x_2^2}_{f_3(x)}; \tag{10}$$
constraints $x_i \in [-5 \quad 5]$ $(i = 1,2)$; global optimizers $x^\star = [-0.0898 \quad 0.7126]^\top$ and $x^\star = [0.0898 \quad -0.7126]^\top$; global optimum $f(x^\star) = -1.0316$.
- `hartman3`: number of agents $N = 4$; $x \in \mathbb{R}^3$; cost function:
$$f(x) = \sum_{i=1}^{4} \underbrace{-c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right)}_{f_i(x)}, \tag{11}$$
with c_i, a_{ij} and p_{ij} being the entries of the matrices:
$$c = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix}, \quad A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}, \tag{12}$$
$$P = \begin{bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4837 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{bmatrix}; \tag{13}$$
constraints $x_i \in [0 \quad 1]$ $(i = 1,2,3)$; global optimizer $x^\star = [0.1140 \quad 0.556 \quad 0.852]^\top$; global optimum $f(x^\star) \simeq -3.8628$.
- `ls`: number of agents $N = 4$; $x \in \mathbb{R}^4$; cost function:
$$f(x) = \sum_{i=1}^{4} \underbrace{\|A_ix - b_i\|^2}_{f_i(x)}, \tag{14}$$
where $A_i \in \mathbb{R}^{100\times4}$ and $b_i \in \mathbb{R}^{100}$. The entries of the matrix A_i are i.i.d. and drawn from a Gaussian $\mathcal{N}(0,1)$ distribution. The vector b_i is generated as follows: $b_i = A_ix_i^\star$, where the

elements of $x_i^\star \in \mathbb{R}^4$ are i.i.d. and drawn from a uniform distribution $\mathcal{U}(-1, 1)$. The elements of b_i and A_i are scaled by a factor $\frac{1}{100}$. The constraints are: $x_i \in [-1 \;\; 1]$ $(i = 1, 2, 3, 4)$; the global optimizer $x^\star$ and corresponding optimum $f(x^\star)$ depend on the random generation of the problem and are computed through centralized convex optimization.

We stress that the global optima are used only to evaluate the quality of the solution obtained by D-GLIS.

In all the experiments the agents communicate over a fixed undirected graph $\mathcal{G}$, generated using an Erdős-Rényi random model (n, p) with $p = 0.3$. The adjaceny matrix $\mathcal{A}$ of the graph is obtained through a Metropolis-Hastings weight model [31].

Algorithm 1 - D-GLIS has been applied on the four aforementioned benchmark problems, running Algorithm 4 - GTAdam as inner distributed solver in `Step 5` and `Step 12`.

In all the experiments the agents start the optimization procedure with an initial local dataset $\mathcal{D}_i$ composed of $M_i = 2n$ points, $i = 1, \dots, N$, generated uniformly at random in the feasible set. For each outer iteration of D-GLIS, the inner solver GTAdam runs for 1000 iterations with an initial stepsize equal to 0.001 in `hartman3`, and equal to 0.01 in the other problems. The value of the exploration hyper-parameter δ is updated by the agents while D-GLIS is running, according to the following heuristic: each agent i uses in `Step 5` a value $\delta_i \triangleq N \left(\max_{y_j \in \mathcal{D}_i} y_j - \min_{y_j \in \mathcal{D}_i} y_j \right)$, which depends on its current dataset $\mathcal{D}_i$.

Each problem is run for 20 independent Monte-Carlo realizations and the performance of D-GLIS is shown in Figure 1 in terms of quantiles. The function values plotted in Figure 1 are obtained after each outer iteration of D-GLIS, by computing $x^* = \arg\min_x \sum_{i=1}^N \hat{f}_i(x)$ in a distributed way, and then evaluating $f(x)$ in $x^\star$. These points have been computed only for the purpose of monitoring the performances of D-GLIS, but, in practice, $x^\star$ is computed only once, in `Step 12`, when the desired maximum number of iterations $T_{\max}$ is reached. Figure 1 shows that on all the tested problems D-GLIS approaches towards the global optimum after less than 80 experiments.

B. Case study: MPC for automated driving vehicles

As a case study, we use D-GLIS to calibrate an MPC controller for automated driving vehicles for lane-keeping and obstacle-avoidance. This case study was originally discussed in in [32], [33] for semi-automatic calibration of MPC parameters through active preference-based optimization.

The test scenario is shown in Figure 2. A Subject Vehicle (SV) is on a one-way horizontal road with two lanes. Unlike the case study tested in [32], [33], we include two Obstacle Vehicles (OVs). Each OV is placed at the center of the corresponding lane and moves forward horizontally at a constant speed. OVs' initial velocities and initial longitudinal positions can vary, depending on the test scenarios. The SV is commanded by an MPC controller to follow the lane, and to avoid OVs (when they are within safety distances) by changing the lane, accelerating, or decelerating. Each calibrator (namely, agent) can prioritize optimization criteria differently and conduct various experiments. The goal is to reach consensus among them without disclosing the specifics of their experiments.

In the following, we provide the system description of the SV (the vehicle under control), the MPC formulation, the control objectives for each agent, and the numerical test performed.

1) System description: A simplified two-degree-of-freedom bicycle model is implemented with the front wheel as the reference point to describe the vehicle kinematics and simulate the experiment:

$$\begin{aligned} \dot{x}_f &= v\cos(\theta + \psi), \\ \dot{w}_f &= v\sin(\theta + \psi), \\ \dot{\theta} &= \frac{v\sin(\psi)}{L}, \end{aligned} \tag{15}$$

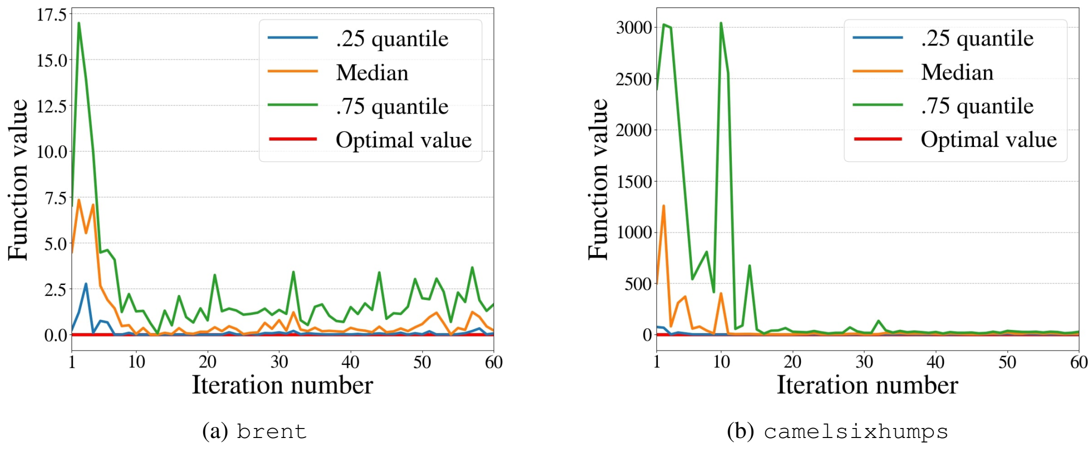
where x_f and w_f (m) are the longitudinal and lateral positions of the SV's front wheel, and θ (rad) is the yaw angle. These three variables define the state vector $s \triangleq [x_f \; w_f \; \theta]^{\mathrm{T}}$. The manipulated variables v and ψ, grouped as $u \triangleq [v \; \psi]^{\mathrm{T}}$, are the SV's velocity v (m/s) and steering angle ψ (rad). L (m) is the length of the SV.

2) MPC formulation: Full state observation is assumed and the control output y coincides with s. The discrete-time state-space model resulting from (15) is

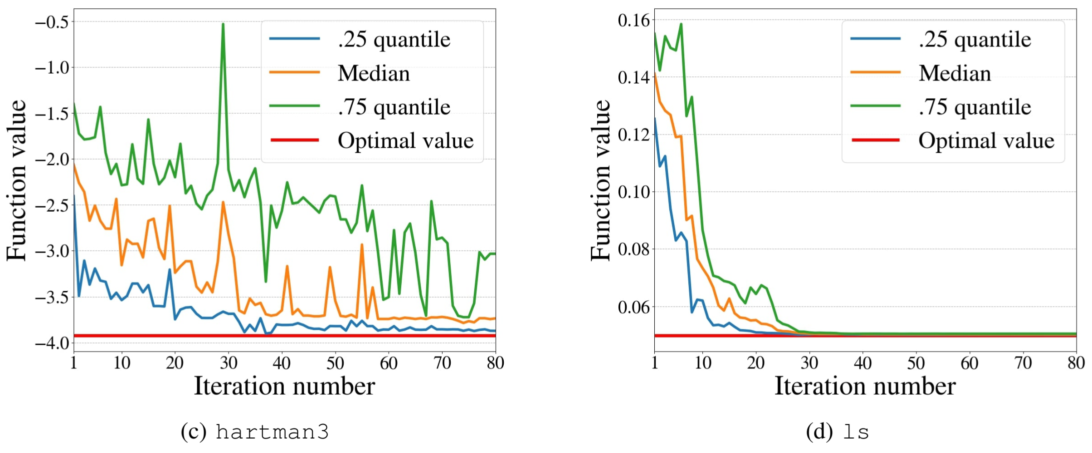
$$\begin{aligned} \tilde{s}_{k+1} =& \begin{bmatrix} 1 & 0 & -\bar{v}_k \sin(\bar{\theta}_k + \bar{\psi}_k) T_s \\ 0 & 1 & \bar{v}_k \cos(\bar{\theta}_k + \bar{\psi}_k) T_s \\ 0 & 0 & 1 \end{bmatrix} \tilde{s}_k \\ &+ \begin{bmatrix} \cos(\bar{\theta}_k + \bar{\psi}_k) T_s & -\bar{v}_k \sin(\bar{\theta}_k + \bar{\psi}_k) T_s \\ \sin(\bar{\theta}_k + \bar{\psi}_k) T_s & \bar{v}_k \cos(\bar{\theta}_k + \bar{\psi}_k) T_s \\ \frac{\sin(\bar{\psi}_k)}{L} T_s & \frac{\bar{v}_k \cos(\bar{\psi}_k)}{L} T_s \end{bmatrix} \tilde{u}_k, \\ \tilde{y}_k =& \tilde{s}_k, \end{aligned} \tag{16}$$

where T_s is the sampling time, subscript k denotes a time step counter, and superscript tilde $\tilde{\cdot}$ and bar $\bar{\cdot}$ are used to indicate deviation and nominal variables, respectively, where $\tilde{\mathrm{x}}_{\mathrm{var}} \triangleq \mathrm{x}_{\mathrm{var}} - \bar{\mathrm{x}}_{\mathrm{var}}$, with $\mathrm{x}_{\mathrm{var}} \in \{s_{k+1}, s_k, v_k, \theta_k, \psi_k, u_k, y_k\}$.

With the prediction model (16), a linear time-varying MPC is designed via a real-time iteration scheme [34], [35]. At each sampling time t, the following quadratic programming problem is solved,



(a) `brent` (b) `camelsixhumps`



(c) `hartman3` (d) `ls`

Fig. 1: Performances of D-GLIS in terms of quantiles on benchmark problems: function value vs. number of iterations.

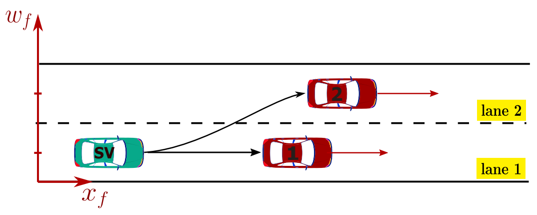


Fig. 2: Test scenario for MPC calibration.

in order to compute the MPC action to be applied:

$$\begin{aligned}\min_{\{u_{t+k|t}\}_{k=0}^{N_u-1}} : & \sum_{k=0}^{N_p-1} \left\|y_{t+k|t} - y_{t+k}^{\mathrm{ref}}\right\|_{Q_y}^2 \\ & + \sum_{k=0}^{N_p-1} \left\|u_{t+k|t} - u_{t+k}^{\mathrm{ref}}\right\|_{Q_u}^2 \\ & + \sum_{k=0}^{N_p-1} \left\|\Delta u_{t+k|t}\right\|_{Q_{\Delta u}}^2 ,\end{aligned} \tag{17}$$

$$\begin{aligned}\text{s.t.}\quad & y_{\min} \leq y_{t+k|t} \leq y_{\max},\ k = 1,\dots,N_p, \\ & u_{\min} \leq u_{t+k|t} \leq u_{\max},\ k = 1,\dots,N_p, \\ & \Delta u_{\min} \leq \Delta u_{t+k|t} \leq \Delta u_{\max},\ k = 1,\dots,N_p, \\ & u_{t+N_u+k|t} = u_{t+N_u|t},\ k = 1,\dots,N_p - N_u,\end{aligned} \tag{18}$$

where Q_y, Q_u, and $Q_{\Delta u}$ are weight matrices for the squared norms, $\Delta u_{t+k|t}$ is defined as $\Delta u_{t+k|t} \triangleq u_{t+k|t} - u_{t+k-1|t}$, y_{ref} and u_{ref} are, respectively, the reference values of control outputs and inputs during the experiments, and N_u and N_p are, respectively, the control and prediction horizon values.

3) Test scenarios and control objectives: In the following we discuss the MPC parameters to be tuned, and specify the constraints both for the input manipulated variables and for the controller outputs of the test scenarios.

We consider the following MPC parameters to calibrate: control and prediction horizons N_u and N_p, respectively; and the diagonal elements of the weight matrix $Q_{\Delta u} \triangleq \begin{bmatrix} q_{u11} & 0 \\ 0 & q_{u22} \end{bmatrix}$. These parameters are tuned within the following intervals: $N_p \in [10 \quad 30]$; N_u is taken as a fraction ϵ_c of N_p, with $\epsilon_c \in [0.1 \quad 1]$; and $\log(q_{u11})$, $\log(q_{u22}) \in [-5 \quad 3]$. The rest of the MPC parameters are fixed and not optimized, with sampling time $T_s = 0.085$ s, $Q_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, and $Q_u = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

The reference value v^{ref} of the manipulated variable v is set to 50 km/h. During the experiments, v can fluctuate within the interval $[1 \quad 90]$ km/h, with its rate of change $\dot{v} \in [-4 \quad 4]$ m/s^2. The steering angle ψ can vary between -45° and 45° at a rate within $[-60 \quad 60]$°/s, with its reference value $\psi^{\text{ref}} = 0°$. As for the control commands: $x_f \in$ [-∞, ∞] m; $w_f \in [-0.6 \quad 3.6]$ m to ensure that SV is within the road; and w_f^{ref} can take the two values 0 m or 3 m (namely, center of lane 1 or lane 2, respectively), depending on which lane the SV is on. The yaw angle θ is constrained to belong to the interval $[-90 \quad 90]$°, and $\theta^{\text{ref}} = 0°$.

The following three objectives have been used to specify how the calibrated MPC controller should direct the SV to maintain lane position, prevent crashes with OVs and guarantee passengers' comfort: $i)$ minimize the variation of the velocity of the SV; $ii)$ minimize the variation of the steering angle of the SV; $iii)$ avoid collision between SV and OVs. The mathematical expressions used to emulate the aforementioned objectives are listed below:

$$\begin{aligned} f_{\text{mult1}} &= \frac{1}{N_{\text{total}}} \sum_{k=1}^{N_{\text{total}}} \left| \frac{v_k - v_k^{\text{ref}}}{v_k^{\text{ref}}} \right|, \\ f_{\text{mult2}} &= \frac{1}{N_{\text{total}}} \sum_{k=1}^{N_{\text{total}}} \left| \frac{\psi_k - \psi_k^{\text{ref}}}{\psi_k^{\text{ref}} + 0.1} \right|, \\ f_{\text{mult3}} &= 1000 \, \mathcal{I}_{\text{collision}}, \end{aligned} \tag{19}$$

where $N_{\text{total}} = 2 \left\lceil \frac{t_{\text{exp}}}{2T_s} \right\rceil$ is the total number of discretization steps throughout the whole experiment, t_{exp} is the experiment duration, and $\mathcal{I}_{\text{collision}}$ denotes an indicator function which takes value 1 if SV and any OV collide, 0 otherwise.

We consider the presence of 4 calibrators (agents) for the case study, each one weighting f_{mult1} and f_{mult2} with a different order of priority. The objective function f_i for agent i (with $i = 1, 2, 3, 4$) is defined below and normalized to the range of $[-2.5 \quad 2.5]$:

$$\begin{aligned} f_1 &= \\ &5(1 - \exp(-0.5 f_{\text{mult1}} - 0.5 f_{\text{mult2}} - f_{\text{mult3}})) - 2.5, \\ f_2 &= \\ &5(1 - \exp(-0.8 f_{\text{mult1}} - 0.2 f_{\text{mult2}} - f_{\text{mult3}})) - 2.5, \\ f_3 &= \\ &5(1 - \exp(-0.3 f_{\text{mult1}} - 0.7 f_{\text{mult2}} - f_{\text{mult3}})) - 2.5, \\ f_4 &= \\ &5(1 - \exp(-0.6 f_{\text{mult1}} - 0.4 f_{\text{mult2}} - f_{\text{mult3}})) - 2.5. \end{aligned} \tag{20}$$

Furthermore, each agent assesses the MPC controller using different initial OV settings (see Table I), i.e., different simulation experiments are performed by the agents. The calibration goal is to reach an agreement among them so that the MPC controller works well under diverse testing circumstances. For this case study, every experiment is simulated for $t_{\text{exp}} = 30$ s.

TABLE I: Initial test conditions of OVs for each agent ($x^0_{f,\text{OV}}$: the initial longitudinal position of the OVs; v^0_{OV}: the initial velocity of the OVs).

Agent	$x^0_{f,\text{OV}}$ [m]		v^0_{OV} [km/hr]	
	OV$_1$	OV$_2$	OV$_1$	OV$_2$
1	10	33	38	40
2	15	17	30	48
3	20	60	40	42
4	9	20	60	45

4) Calibration process: The calibration of the MPC parameters N_p, ϵ_c, $\log(q_{u11})$, and $\log(q_{u22})$ have been performed by running D-GLIS with the following configuration:

- the i-th agent (with $i = 1, 2, 3, 4$) only knows the local function $f_i(x)$, defined above.
- the agents communicate over a fixed undirected graph $\mathcal{G}$, generated using an Erdős-Rényi random model (n, p) with $p = 0.3$. The adjaceny matrix $\mathcal{A}$ of the graph is obtained through the Metropolis-Hastings weight model.
- Algorithm 4 - GTAdam has been used as inner distributed solver in `Step 5` and `Step 12`.
- the i-th agent (with $i = 1, 2, 3, 4$) has an initial local dataset $\mathcal{D}_i$ composed of $M_i = 2$ feasible points generated uniformly at random.
- for each iteration of D-GLIS, the inner solver GTAdam runs for 1000 iterations with an initial stepsize equal to 0.001.
- the value of the hyper-parameter δ is updated while D-GLIS is running, according to the same heuristic described in Section V-A.
- D-GLIS terminates after $T_{\max} = 80$ iterations.

5) Results: D-GLIS provides MPC parameters with satisfactory performance with 2 initial random experiments and 14 active learning experiments for each agent (64 experiments in total). The optimized parameters of [ϵ_c, N_p, $\log(q_{u11})$, $\log(q_{u22})$] are $x^\star = [0.10, 20, \text{-}4.25, \text{-}5]^\top$.

We show how the performance of the MPC controller is enhanced by using the optimal MPC parameters by comparing it to its initial performance by using instead one of the initial random MPC parameters $\bar{x} = [0.86 \ 27 \ 1.8 \ 0.23]^\top$, retrieved from the dataset $\mathcal{D}_3$. This MPC parameter vector $\bar{x}$ is used to simulate the experiments of all four agents, only with the purpose of comparing performance results.

The objective function evaluations for the experiments simulated by the four agents with the initial and optimal MPC parameters are, respectively, $\{f_1(\bar{x}) = -0.9079,\ f_2(\bar{x}) = -0.6444,\ f_3(\bar{x}) = -1.426,\ f_4(\bar{x}) = -0.9181\}$ and $\{f_1(x^\star) = -2.249,\ f_2(x^\star) = -2.384,\ f_3(x^\star) = -2.162,\ f_4(x^\star) = -2.292\}$. The MPC controller with the optimal parameters thus obtains lower objective functions for the experiments of every agent. In contrast, the MPC controller with the random initial parameters performs more inconsistently among experiments of different agents and leads to higher function evaluations. More specifically, the MPC controller with the initial parameters performs poorly for the experiments by Agents 1, 2, and 4 and mediocrely for the experiments by Agent 3. These observations are also highlighted in Figure 3, where the evolution of the two manipulated variables (velocity v and steering angle ψ) for each agent is plotted. From Figure 3, it is also observed that, compared to the MPC controller with the optimal parameters, the MPC controller with the initial parameters generally exhibits more aggressive and frequent fluctuations in both manipulated variables throughout the experiment duration, whose values often also deviate from the reference.

VI. Conclusions

This paper has proposed D-GLIS, an algorithm to solve cooperatively, over a distributed network of agents, global optimization problems where the cost function is separable and expensive to evaluate, subject possibly to global constraints (known and inexpensive to evaluate). The proposed scheme, contrarily to many other approaches for black-box optimization, e.g., Bayesian optimization, is driven by deterministic arguments (the IDW function), which has the purpose of encouraging the investigation of unexplored regions of the feasible space. Differently from its predecessor GLIS, D-GLIS is a distributed scheme, meaning that a network of agents cooperate to solve the common optimization problem through distributed experiments, exchanging among themselves only fundamental information about the minimization procedure.

Current research directions are devoted to: 1) the extension of D-GLIS to problems where the constraints can also be local/private and, furthermore, expensive to evaluate; and 2) developing a preference-based version of the proposed approach, always distributed, where the agents can not evaluate the cost function, but it is only possible to obtain preferences, as such "this is better than that", between two candidate points.

References

[1] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv:1012.2599*, 2010.

[2] Hans-Martin Gutmann, "A radial basis function method for global optimization," *Journal of Global Optimization*, vol. 19, pp. 201–2227, 2001.

[3] Rommel G. Regis and Christine A. Shoemaker, "Constrained global optimization of expensive black box functions using radial basis functions," *Journal of Global optimization*, vol. 31, no. 1, pp. 153–171, 2005.

[4] Cédric Malherbe and Nicolas Vayatis, "Global optimization of lipschitz functions," *International Conference on Machine Learning*, pp. 2314–2323, 2017.

[5] A. Bemporad, "Global optimization via inverse distance weighting and radial basis functions," *Computational Optimization and Applications*, vol. 77, pp. 571–595, 2020.

[6] Lorenzo Sabug Jr., Fredy Ruiz, and Lorenzo Fagiano, "SMGO: a set membership approach to data-driven global optimization," *Automatica*, vol. 133, pp. 109890, 2021.

[7] Lindsay Bassman, Pankaj Rajak, Rajiv K. Kalia, Aiichiro Nakano, Fei Sha, Jifeng Sun, David J. Singh, Muratahan Aykol, Patrick Huck, Kristin Persson, and Priya Vashishta, "Active learning for accelerated design of layered materials," *npj Computational Materials*, vol. 4, no. 1, pp. 1–9, 2018.

[8] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato, "Constrained bayesian optimization for automatic chemical design using variational autoencoders," *Chemical science*, vol. 11, no. 2, pp. 577–586, 2020.

[9] Loris Roveda, Marco Forgione, and Dario Piga, "Robot control parameters auto-tuning in trajectory tracking applications," *Control Engineering Practice*, vol. 101, pp. 104488, 2020.

[10] Loris Roveda, Mauro Magni, Martina Cantoni, Dario Piga, and Giuseppe Bucca, "Human–robot collaboration in sensorless assembly task learning enhanced by uncertainties adaptation via bayesian optimization," *Robotics and Autonomous Systems*, vol. 136, pp. 103711, 2021.

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[12] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas, "Bayesian optimization in alphago," *arXiv preprint arXiv:1812.06855*, 2018.

[13] A. Nedić and J.s Liu, "Distributed optimization for control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 77–103, 2018.

[14] Ying Sun, Marie Maros, Gesualdo Scutari, and Guang Cheng, "High-dimensional inference over networks: Linear convergence and statistical guarantees," *arXiv preprint arXiv:2201.08507*, 2022.

[15] L. Cannelli, F. Facchinei, G. Scutari, and V. Kungurtsev, "Asynchronous optimization over graphs: Linear convergence under error bound conditions," *IEEE Transactions on Automatic Control*, vol. 66, no. 10, pp. 4604–4619, 2020.

[16] Lucian Busoniu, Robert Babuska, and Bart De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[17] J. Choi, S. Oh, and R. Horowitz, "Distributed learning and cooperative control for multi-agent systems," *Automatica*, vol. 45, no. 12, pp. 2802–2814, 2009.

[18] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *Advances in neural information processing systems*, vol. 29, 2016.

[19] Pengcheng Shen, Chunguang Li, and Zhaoyang Zhang, "Distributed active learning," *IEEE Access*, vol. 4, pp. 2572–2579, 2016.

[20] Paolo Di Lorenzo and Gesualdo Scutari, "Next: In-network nonconvex optimization," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 120–136, 2016.

[21] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

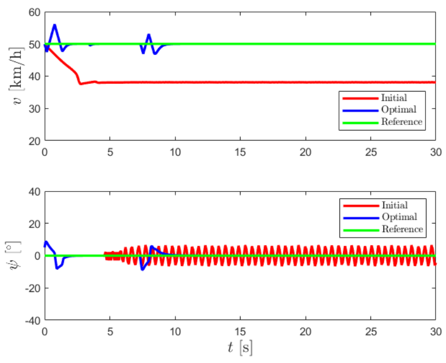
[22] Dale B. McDonald, Walter J. Grantham, Wayne L. Tabor, and Michael J. Murphy, "Global and local optimization using radial basis function response surface models," *Applied Mathematical Modelling*, vol. 31, no. 10, pp. 2095–2110, 2007.

[23] A. Bemporad and D. Piga, "Global optimization based on active preference learning with radial basis functions," *Machine Learning*, vol. 110, pp. 417–448, 2021.

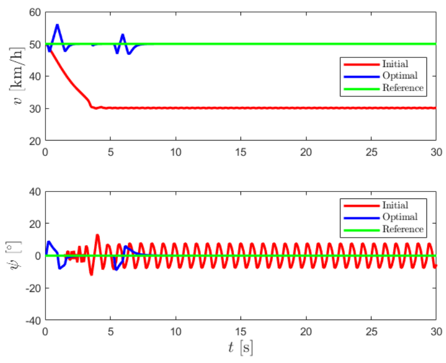
[24] G. Carnevale, F. Farina, I. Notarnicola, and G. Notarstefano, "Distributed online optimization via gradient tracking with adaptive momentum," *arXiv:2009.01745*, 2020.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

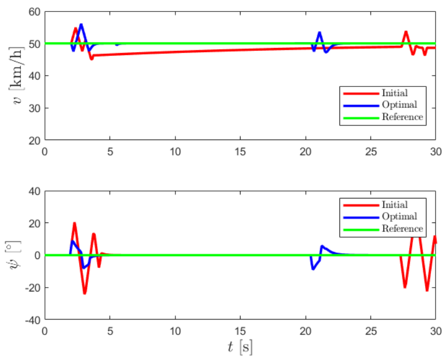
[26] Seunghye Lee, Jingwan Ha, Mehriniso Zokhirova, Hyeonjoon Moon, and Jaehong Lee, "Background information



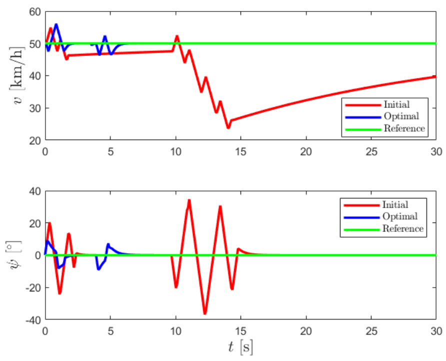
(a) `Agent 1`



(b) `Agent 2`



(c) `Agent 3`



(d) `Agent 4`

Fig. 3: SV control performances obtained by the MPC controller with initial parameters $\bar{x}$ (red curves) and optimal parameters $x^\star$ (blue curves), tested on four different simulation experiments by the agents. Each subfigure includes two subplots showing the evolution of velocity v (top) and steering angle ψ (bottom) of the SV with respect to the simulation time t.

of deep learning for structural engineering," *Archives of Computational Methods in Engineering*, vol. 25, no. 1, pp. 121–129, 2018.

[27] Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava, "Fast and scalable bayesian deep learning by weight-perturbation in adam," *International Conference on Machine Learning*, pp. 2611–2620, 2018.

[28] M. Zhu and S. Martínez, "Discrete-time dynamic average consensus," *Automatica*, vol. 46, no. 2, pp. 322–329, 2010.

[29] F. Farina, A. Camisa, A. Testa, I. Notarnicola, and G. Notarstefano, "Disropt: a python framework for distributed optimization," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 2666–2671, 2020.

[30] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[31] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," *Fourth International Symposium on Information Processing in Sensor Networks*, pp. 63–70, 2005.

[32] M. Zhu, A. Bemporad, and D. Piga, "Preference-based MPC calibration," *European Control Conference (ECC)*, pp. 638–645, 2021.

[33] M. Zhu, D. Piga, and A. Bemporad, "C-GLISp: Preference-based global optimization under unknown constraints with applications to controller calibration," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 2176–2187, 2022.

[34] Moritz Diehl, Hans G. Bock, and Johannes P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[35] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear MPC: bridging the gap via the real-time iteration," *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020.